1. A view of the VSO being currently implemented

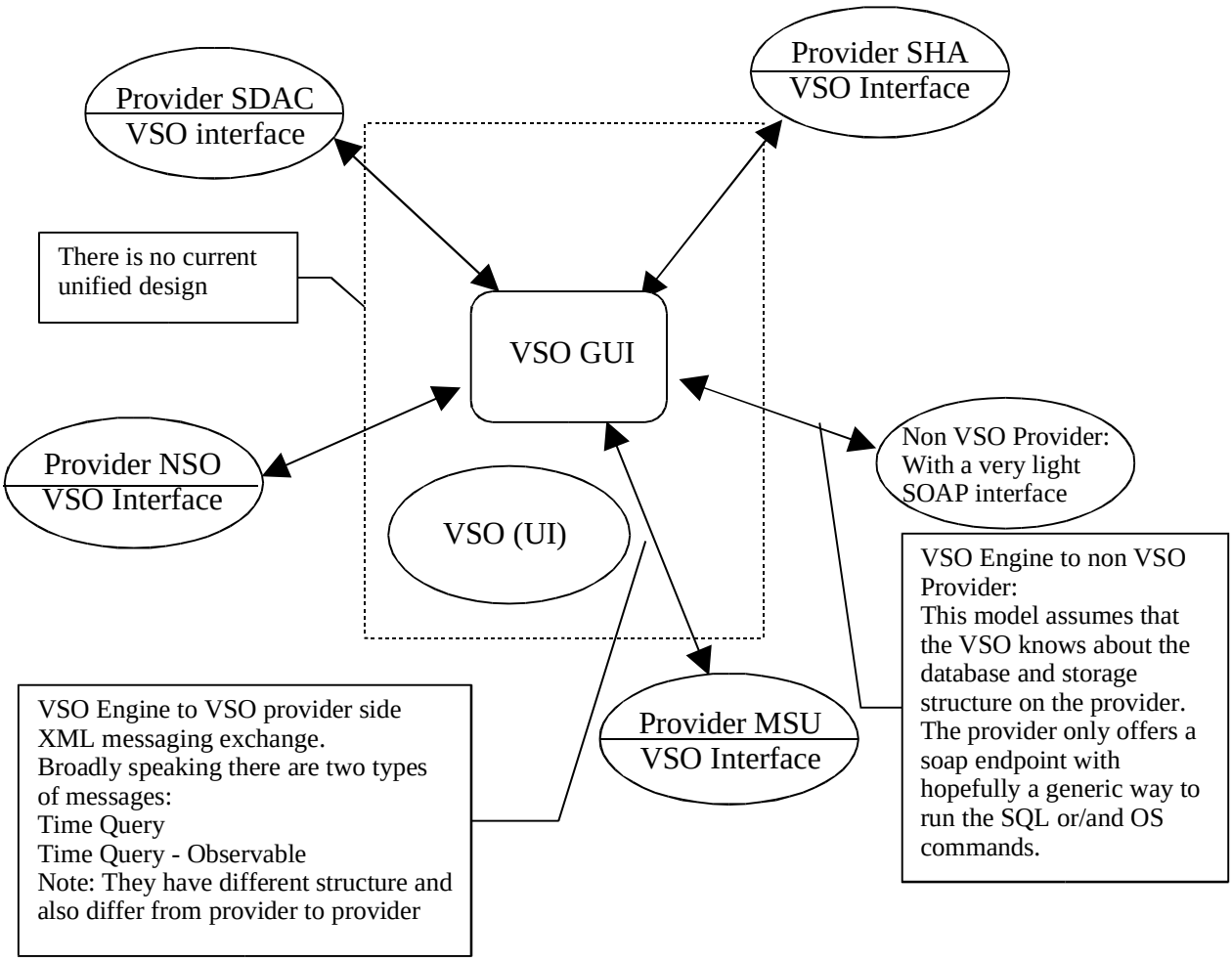

Figure 1

A "VSO Provider" can be represented as follows:

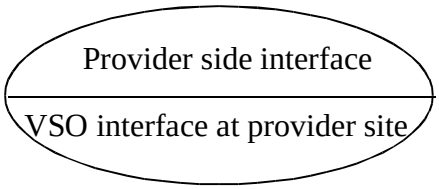

Figure 2

a. VSO ENGINE

Currently there is no unify implementation of it.

There are two attemps:

i.Stanford VSO GUI

It is based mostly on a "VSO Provider" model

The GUI is currently hard wired to two types of soap calls.
A query time soap call
Different calls are made to each one of the providers
E.g.
Three calls for NSO
One for MSU
One for SDAC
One for SHA
A Query time-observable soap call
The same call message is used for all providers
The VSO GUI is able to perform a "JOIN" query. That's it to take the output of one call as the input for the other, and display the output of both accordingly.

ii.MSU VSO UI

It is based on a "non VSO Provider" model. (There is no GUI interface yet)

.- It provides some generic way to call using SOAP.
.- The XML message structure is different from the Stanford implementation.
.- The complexity of the provider lies at the VSO side
.- Mostly intended for SQL queries
.- MSU and SDAC are current implementations

b. A tentative work ahead list from a technical perspective in order to achieve point The VSO end product:

.-Define how the data will be structure within the VSO (VSO data model) and how would it be represented. E.g. XML, perl hashes, tag pair values …
.- Decide what model will be centered to the VSO architecture regarding to the providers
.- Define a flexible UI and some rudimentary version of the VSO ENGINE see Figure 3.
E.g.
SOAP calls:
timeQuery (obsstart, obsend)
Input XML:

```
<timeQuery>
    <OBSSTART>YYYY-MM-DD hh:mm:ss TZ</OBSTART>
    <OBSEND> YYYY-MM-DD hh:mm:ss TZ</OBSTART>
</timeQuery>
```

Output XML: …

obsQuery(obsstart,obsend, observable, instrument)
…
.- Modify the GUI to use the VSO UI architecture
.- To instantiate the VSO engine in several locations: E.g. Stanford, NSO, MSU
.- Define a unified message exchange for the different VSO parts where posible.

2. The VSO end product

This is how I understand the VSO people sees the VSO. Some of it might be just a product of my imagination while others might be things I picked up from conversations and reading in VSO WEB pages.

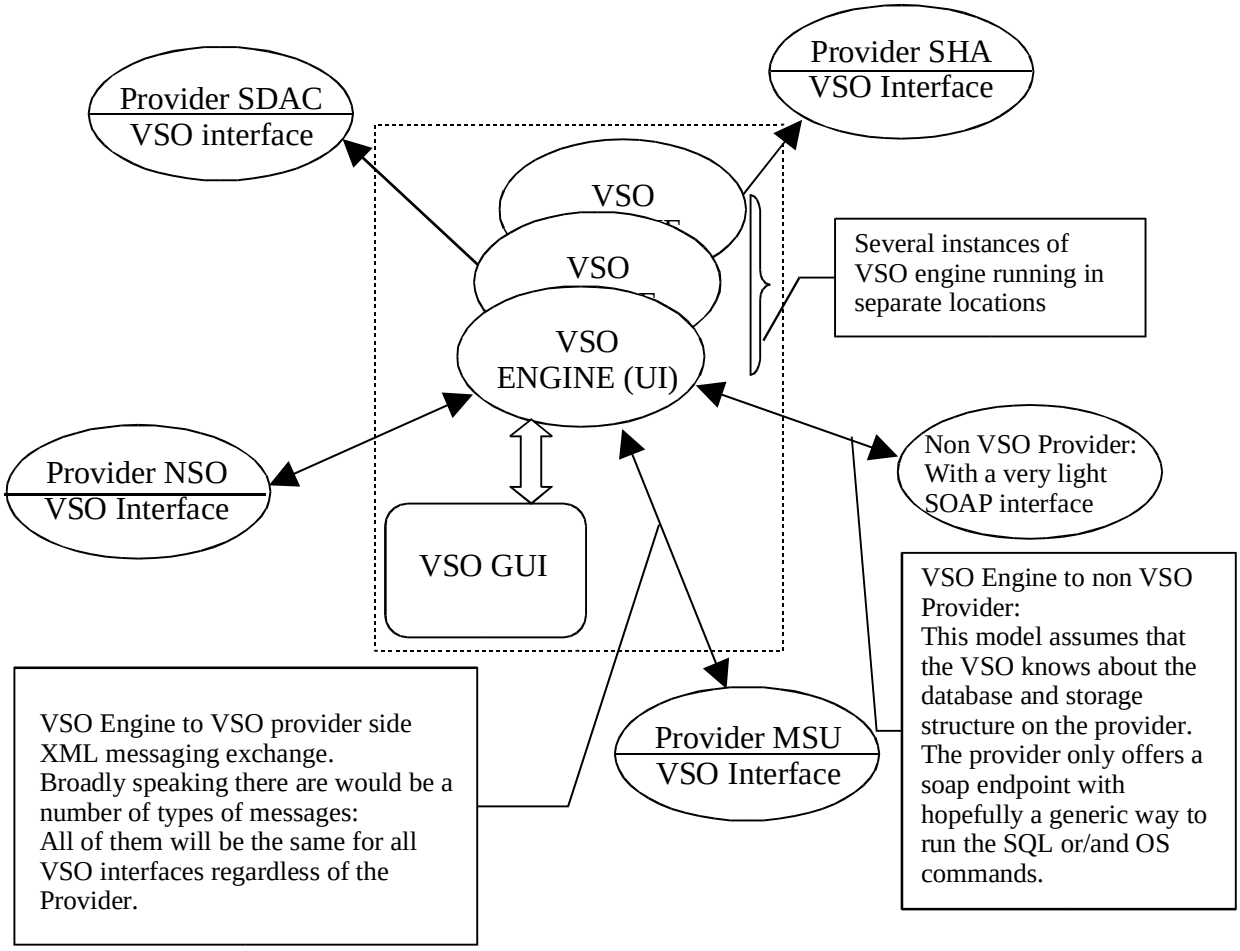

Figure 3

A view of the VSO Engine

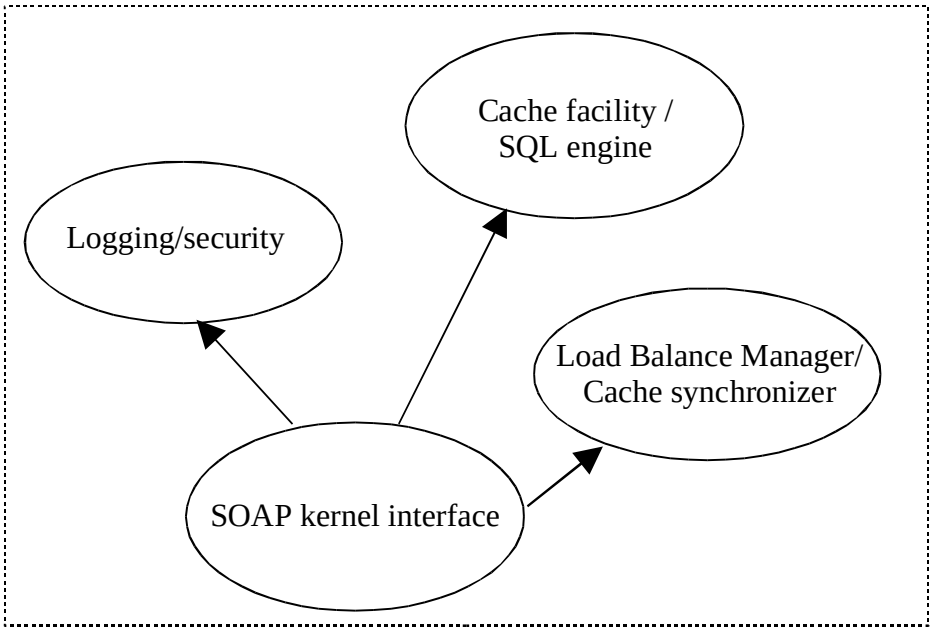

Figure 4

The VSO Engine will be instantiated in several locations. The "Load Balance Manager and Cache synchronizer" will ensure synchronization and optimal performance amond engines.

a. The UI

The VSO engine should provide the UI. The VSO UI can be similar to that of a UNIX system or/and a relational database

I.e.
The VSO engine could implement a mechanism so a client(user) can :

execute a query given a XML message
E.g.
vsoquery(xml)

or/and

A number of wrapper functions that hide the xml complexities from the user

getdatadescr(obsstart, obsend, <keywork>, <value>, <options>, <value>, …)
getdata(description_id) # An interface to retrieve the data.

More particularized ones (dependent to language implementation C, Java, Perl, etc)
vsoquerytime(obsstart,obsend)
vsoqueryobsevable(obsstart,obsend, observable)
vsoqueryinstrument(obstart, obsend, intrument)
vsojoin(obsstart,obsend,<keyword>), where <keyword> is observable, instrument etc

….

Or a OO version
vsoquery(obstart,obsend)
vsoquery(obsstart,obsend, <keywork>)

Implement a virtual SQL engine (It can be a relational database)
where commands like
"select observable, obsstart, obsend, instrument from VSO_PROVIDER where …"

or

"select observable, obsstart, obsend, instrument from VSO_PROVIDER V1, VSO_PROVIDER V2 where V1.obstart = "…." and V1.obstart = " …." and V1.obstart = V2.obstart and V1.observable = " …" and V1.observable = V2.observable"

can be understood by the VSO Engine

ETC …

All these commands or interfaces will run from the user side, although effectively will be executed at the VSO Engine.

All the above will be supported by "kernel" like functions at the VSO Engine side that the user won't have access to. Internally the argument to these functions will be solely XML, input and output.

b. Cache system:

The VSO engine will enquiry and get an up-to-date information on the provider holdings.
This caching mechanism can be made either by one big request or by incremental requests.
E.g.

exportdata(<provider>, <date>) # The VSO provider will give a detail data export of their holdings.
That information can be cached a the VSO Engine side for fast processing and SQL processing.

c. The GUI

The GUI will be a completely separate entitiy and will use the user interface in other to retrieve the data.